CALL PREEMPTION IN COMMUNICATION NETWORKS*

(Extended Abstract)

JUAN A. GARAY and INDER S. GOPAL

IBM T. J. Watson Research Center
P. O. Box 704
Yorktown Heights, NY 10598, USA
{garay, gopal}@watson.ibm.com

Abstract

The technique of preempting ongoing calls in order to accommodate new calls of greater "value" has been used in many existing networks. In this paper we investigate some problems that relate to making the best decision on which (if any) call to preempt. We provide some results on the computational complexity of these problems and on how well simple heuristic procedures can approximate the optimal strategy.

1 Introduction

It is generally accepted that future high speed networks will be based on packet switching technology. Various alternatives (e.g., ATM [ATM], PARIS [CG]) have been proposed and are in various stages of development. Underlying all such design alternatives is the assumption that high speed networks will have to support a wide variety of traffic types. Each of these traffic types will have very different requirements in terms of delay, throughput, and packet loss. It is likely that many of these requirements will only be met if bandwidth can be reserved for individual connections, in accordance with their requirement.

Numerous previous works have focussed on the problem of bandwidth reservation and management. The objective has been to maximize the usage of the network facilities, while minimizing the probability that a particular connection will be denied access to the network or "blocked." The problem remains one of the most actively studied open issues in the area of high speed networks. The fact that calls have different bandwidth and the complications introduced by the availability of alternate paths within the network make the problem extremely difficult.

The nature of the problem is such that it is always possible to "second guess" decisions made in the past. In other words, a decision made previously to accept a connection (we use the term *call* or *connection* interchangeably) may have been wrong because it caused a subsequent more "valuable" call to be rejected. For example, an accepted call may have used up capacity from the network, thereby causing a call of much larger capacity (and consequently larger revenue to the network) to be rejected. (We use *capacity* and *bandwidth* interchangeably.) Thus, the *on-line* nature of the problem (the fact that decisions are to be made when calls arrive into the system without knowledge of future arrivals) leads to significantly lower "efficiency" than would have been possible with

*Appeared in *Proc. INFOCOM '92*, pp. 1043-1050, Florence Italy, May 1992.

full *off-line* knowledge of the entire pattern of call arrivals.

This aspect of the problem leads naturally to the investigation of the issue of *preemption*. In other words, if previous decisions were incorrect, it may pay to attempt to rectify them by preempting or removing a call in progress from the network. For example, if a high capacity call is blocked because of capacity used up by a low capacity call, it may be worthwhile to preempt the low capacity call and accept the high capacity call even at the risk of interrupting a call in progress.

The value of a call is not determined simply by its capacity. In certain cases, there may be an ordering of calls determined by some predefined "priority." For example, calls made from the headquarters of a company may be deemed to be more important than calls made from a remote branch office. Also, geographical distance may be a factor. Cross-country calls may be deemed more valuable than calls made locally as the potential revenue from them is higher. The holding time of a call may also be an important factor in determining its value. If a call is likely to remain in the system for an extended period, it is more valuable than a call that is only for a brief period. Of course, frequently the holding time is not known at the time a call arrives and therefore it may be difficult to consider this parameter.

Call preemption is used in many real systems. For example, in many T1 and T3 bandwidth manager products (e.g., NET IDNX), call preemption is an important method used to ensure that call routing is done efficiently. Call preemption is also proposed as a feature of the *plaNET* network being deployed in the *AURORA* testbed [CDF+].[1] To the best of our knowledge, no analytical study of call preemption has been published.

In this paper, we remedy this deficiency somewhat by formulating and investigating some aspects of the call preemption problem. Our study clearly does not comprise a complete investigation of the problem, rather it represents some selected problems that we believe are at the crux of call preemption. Through our involvement with the *AURORA* testbed we plan to implement and experimentally test out some alternatives. Thus this paper can be viewed as one aspect of our overall investigation. We plan to conduct the experimental study in close connection with the theoretical work and to publish a more complete view of the problem in a subsequent paper.

The remainder of the paper is structured as follows. In Section 2, we present the general formulation of the single-step preemption problem and show that in that form the problem is computationally intractable. We then show that heuristics exist which can achieve a performance that is within a logarithmic factor of optimal. In Section 3 we investigate an issue that is not commonly studied in such systems, i.e. the loss of efficiency we suffer by the *on-line* nature of the problem. In particular, if we compared the performance of the on-line algorithm with an off-line algorithm that obtained all the information about future call arrivals, how close would we be? Intuition suggests that a knowledge of future arrivals can help enormously. In this paper, however, we show that on-line algorithms can, in some circumstances perform as well as off-line algorithms.

2 The Single-Step Preemption Problem

We assume that the network under consideration is a general graph, $G = (V, E)$. The routing for each call is defined by a process that is outside of the scope of the call preemption algorithm. Thus calls "arrive" with some predefined bandwidth or capacity and a predefined route. We use the notation c_i to indicate the ith call arrival. Each call c_i is determined by the tuple $< a_i, b_i, r_i, d_i >$, where a_i and d_i are the call's arrival and holding times, respectively (not necessarily known in advance), b_i is the bandwidth, and r_i is the route. r_i is an ordered set of edges $\{e_{i_1}, e_{i_2}, ...\}$ drawn from the set E with the property that r_i comprises a path in G. For simplicity, we assume bidirectional edges and capacity assignments.

Each edge, $e \in E$, has a bandwidth $B(e)$ associated with it. If a call is accepted into the system, it must have the property that for each edge used by the call the sum of the bandwidth of all the calls that share the edge must be less than the capacity of the edge. (This includes the capacity of

[1] *AURORA* is a Gigabit/second wide-area networking test-bed sponsored by NSF and DARPA under the umbrella of the Corporation for National Research Initiatives.

the newly added call.) Otherwise, the call must be rejected or some of the preexisting calls preempted.

The *single-step call preemption* problem ($SSCP$) is to minimize the "disruption" to the existing calls if preemption is required. In other words, the objective is to find some set of existing calls, P, such that after every call from this set is removed from the network, the resultant network has the capacity to accept the new call. Several different metrics can be defined on the set P to capture the notion of minimal cost:

1. the cardinality of P;
2. the total bandwidth of the calls in P;
3. the total network bandwidth of the calls in P,

where the network bandwidth of a call is the bandwidth of the call times the number of hops in the route of the call. We will refer to these versions of the $SSCP$ problem as $CP1$, $CP2$, and $CP3$, respectively.

We show first that all versions of the problem are $\mathcal{NP}$-complete. We refer to the well-known text by Garey and Johnson for a definition of terminology [GJ].

Proposition 2.1 *$CP1$, the version of SSCP based on the number of preempted calls, is $\mathcal{NP}$-complete.*

Proof: Clearly the problem is in $\mathcal{NP}$. To prove that the problem is $\mathcal{NP}$-complete, we select a well known $\mathcal{NP}$-complete problem, the Minimal Set Cover problem (MSC) [GJ]. We show that for any instance of MSC it is possible to construct in polynomial time a corresponding instance of $CP1$ such that the instance of $CP1$ is solvable in polynomial time if and only if the original instance of MSC is solvable in polynomial time.

To restate the MSC problem here:

Given is a collection $\mathcal{A}$ of subsets of a finite set S, and a positive integer $K \leq |\mathcal{A}|$.

Question: Does there exist a cover for S of size K or less, i.e., a subset $\mathcal{A}' \subseteq \mathcal{A}$ with $|\mathcal{A}'| \leq \mathcal{K}$, such that every element of S belongs to at least one member of $\mathcal{A}'$?

Consider an instance of MSC. We now construct an instance of $CP1$. The graph G upon which this problem is defined consists of a ladder-type structure with two layers of aligned nodes connected by horizontal and vertical edges. Each horizontal edge in the upper part of the ladder and the vertical edges are assigned capacity M, where M is some very large integer. The number of horizontal edges in the lower (and upper) part of the graph is $L = |S|$. Let the lower horizontal edges be labelled $l_1, l_2, ..., l_L$, and the vertical edges $v_1, ..., v_L + 1$. We order the members of S in an arbitrary fashion $s_1, s_2, ..., s_L$ and make a correspondence between s_i and the edge l_i for all $1 \leq i \leq L$. The subsets of S in the collection $\mathcal{A}$ will be made to represent the ongoing calls. Consider a member, A, of the collection $\mathcal{A}$. Order the members of A in the order in which they appear in the ordering chosen for S. Let the members be $s_{A1}, s_{A2}, s_{A3}, ...$ We try to create a call c which includes only the edges $l_{A1}, l_{A2}, ...$ This is easily done by choosing a route for c that alternates appropriately between upper and lower segments of the ladder, using the vertical segments of the ladder to traverse between upper and lower segments. For example, if $u_1, u_2, ...$ denoted the upper horizontal edges, then the route selected for the above example would be $u_1, u_2, ..., u_{A1-1}, v_{A1}, l_{A1}, v_{A1+1}, u_{A1+1}, ...$

In this manner, we create a collection of calls, C, each call containing a subset of the lower horizontal edges. To complete the instance of $CP1$, we assign to each of the calls a capacity of 1 and to each of the lower edges a capacity which is equal to the number of calls that cross it. Finally, the new call that arrives is also given capacity 1, and is assigned a route $l_1, l_2, l_3, ...l_L$.

We now show that this instance of $CP1$ is solvable in polynomial time if and only if the original instance or MSC is so solvable. Consider the "only if" part. In a solution of the $CP1$ instance, the set of preempted calls must contain at least one call for each edge in the lower portion of the ladder because every edge is full to capacity. Using the one-to-one correspondence between calls and subsets, the collection of preempted calls defines a collection of subsets which form a set cover for S. The minimal number of preempted calls forms a minimal set cover. Thus a solution of $CP1$ implies a solution of MCP completing the proof of the "only if" part. The proof of the "if" part follows in a similar fashion. ☐

Proposition 2.2 *$CP2$, the version of SSCP based on the total call bandwidth, and $CP3$, the version based on total network bandwidth of the calls preempted. are $\mathcal{NP}$-complete*

Algorithm $H1$(new call c);
{let $r = \{e_1, e_2, ..., e_n\}$ be the portion of c's route without capacity to accomodate c}
{let $C = \{c_1, c_2, ..., c_p\}$ be the set of existing calls on r; c_j's route is denoted by r_j}
$\pi := r$;
$P := \emptyset$; { the set of calls on r to be preempted to accommodate c}
while $\pi \neq \emptyset$ **do**
{*} let $c_i \in C$ be call s.t. $|\pi \cap r_i|$ is maximal;
$\pi := \pi - (\pi \cap r_i)$;
$C := C - \{c_i\}$;
$P := P \cup \{c_i\}$
endwhile;
return(P).

Figure 1: A heuristic to minimize the number of preempted calls.

Proof: [Sketch] The construction in the previous Proposition had all calls requiring the same bandwidth (1 unit). Thus the claim for $CP2$ trivially follows. The proof for $CP3$ follows from the observation that that MSC remains $\mathcal{NP}$-complete even if all subsets contain exactly three members. □

2.1 Simple Preemption Heuristics

Having established the computational difficulty of the versions of *SSCP*, in this subsection we present simple heuristic procedures that achieve a performance that is within a logarithmic factor of optimal. We measure the quality of the performance of heuristic H for problem P, $P \in \{CP1, CP2, CP3\}$, by the worst-case ratio between the behavior of H and the best possible achievable solution. More formally, given n, the length of the arriving call's route, we define $R[P,H](n) = \mathcal{S}(H(P,n))/\mathcal{O}(P,n)$, where $\mathcal{S}(H(P,n))$ is the (measure of the) worst (maximal) solution produced by H on call length n, and $\mathcal{O}(P,n)$ is the best (minimal) solution to P. We will often delete n and talk about $R[P,H]$. We divide our presentation into two cases: when the bandwidth of the calls is equal (comparable), and otherwise.

2.1.1 Equal (Comparable) Bandwidth

Observe that, under the assumption, optimization metrics (versions of *SSCP*) 1 and 2 from the previous section collapse. We will continue to refer to this version of the problem as $CP1$. We now describe two heuristic procedures $H1$ and $H2$ for the two versions of the problem, $CP1$ and $CP3$. Both heuristics have a $O(n^2 \log n)$ running time, where n is the length of the arriving call's route. ForFor simplicity, we assume that the capacity requirement of each call is 1. The two heuristics can be seen as counterparts of the approximation algorithms for versions of the Set Cover problem that try to find the subcover with fewest sets, and with the least overlapping, respectively (see [D]).

Heuristic $H1$ is shown in Figure 1. The algorithm iterates until there is "room" for the arriving call in every link of the call's route, selecting for preemption at each iteration step the existing call whose route overlaps the most with the portion of the new call route that remains uncovered.

Proposition 2.3 $R[CP1, H1] = O(\log n)$*, where* n *is the length of the new call's route.*

It is easy to see that the bound is attainable: consider the case $n = d \cdot 2^k$ links, for some constants d and k, and the set C of existing calls consists of the following configuration of calls:

- d disjoint calls occupying 2^k links each;
- one call occupying $d \cdot 2^k - 1$ links that overlap with $2^k - 1$ of each of the previous calls;
- one call occupying $d \cdot 2^k - 2$ links that overlap with $2^k - 2$ of each of the d first calls;

...

- $d - 1$ calls occupying d links each, and each call overlapping in one link with each of the d first calls.

The optimal strategy consists of preempting the first d calls, while $H1$ might preempt all the remaining calls from C. We refer the reader to [D] for details on the upper bound part. We also note that the bound holds for n sufficiently large; we expect the heuristic to perform better than the established bound in most practical situations.

Algorithm $H3$(new call c);
  {let $r = \{e_1, e_2, ..., e_n\}$ be the portion of c's route without enough capacity to accomodate c; ρ_j is e_j's residual capacity}
  {let $C = \{c_1, c_2, ..., c_p\}$ be the set of existing calls on r; $c_i =< b_i, r_i >$}
  for $e_j \in r$ **do**
    $w_j := \lceil b - \rho_j \rceil^+$;
  $P := \emptyset$; { the set of calls on r to be preempted to accommodate c }
  repeat
{*}   let $c_i \in C$ be call s.t. $\sum_{e_k \in r_i} \min(w_k, b_i)$ is maximal;
    $C := C - \{c_i\}$;
    $P := P \cup \{c_i\}$
    for $e_k \in r_i$ **do**
      $\rho_k := \rho_k + b_i$;
      $w_k := \lceil b - \rho_k \rceil^+$;
  until $\rho_j \geq b$, $1 \leq j \leq n$;
  return(P).

Figure 2: A heuristic to preempt minimizing total bandwidth.

Our second heuristic, $H2$, for problem $CP3$ (i.e., a heuristic to preempt minimizing the total network bandwidth), is basically obtained from Figure 1 by replacing line {*} by:

let $c_i \in C$ be call s.t. $|r_i - \pi| \ / \ |r_i \cap \pi|$ is minimal;

We note that the procedure concerns the optimization of the total network bandwidth on the arriving call's route. As in $H1$, the algorithms proceeds greedily, but this time the selection of the next call to preempt also takes into account the overlap—and corresponding capacity loss—between the call and the portion of the route already taken care of.

Proposition 2.4 $R[CP3, H2] = O(\log n)$, *where* n *is the length of the new call's route.*

The lower bound follows from an example much like the one for Proposition 2.3 (see [D]). We leave the complete proof for the full version of the paper.

2.1.2 Unequal Bandwidth

In this subsection we present two other heuristics, $H3$ and $H4$, which are, effectively, generalizations of $H1$ and $H2$, respectively. Each call c_i has now a bandwidth b_i associated with it; we let $\beta = \max\{b_i : c_i \in C\}$. For each link j, $1 \leq j \leq n$, we define ρ_j as the *residual* capacity of link j, i.e., the amount of capacity available on the link. We also define the capacity (still) *wanted* by arriving call c as $w_j = \lceil b - \rho_j \rceil^+$, $1 \leq j \leq n$.

We now present heuristic $H3$ for problems $CP1$ and $CP2$ (recall that $CP2$ concerns the minimization of the total bandwidth of the preempted calls). $H3$ is shown in Figure 2. As can be seen from line {*} in the code, the algorithm proceeds by making at each step as much "room" as possible for new call c. We are able to assert the following:

Proposition 2.5 $R[CP2, H3] = O(\beta \cdot \log n)$, *where* n *is the length of the new call's route, and* β *is the maximal bandwidth of an existing call.*

Proof: [Sketch] We call the current problem P_β, and transform it as follows: we replace each existing call $c_i \in C$ by b_i calls, each requiring capacity 1; we call this new problem P_1. We refer to the original call c_i as the "parent" of the new b_i calls produced by the transformation. By running $H1$ on this new scenario, we know (Proposition 2.3) that

$$\mathcal{S}(H1(P_1))/\mathcal{O}(P_1) = O(\log n) \tag{1}$$

holds. It is also easy to see that

$$\mathcal{O}(P_1) \leq \mathcal{O}(P_\beta). \tag{2}$$

We now define the "sloppier" (with respect to $H3$) heuristic that performs as $H1$, but when a selection is made, all the calls comprised by the parent call are removed from C. We call this heuristic $H1_\beta$. By using (1) and (2):

$$\mathcal{S}(H1_\beta(P_\beta)) \leq \beta \cdot \mathcal{S}(H1(P_1)) \leq O(\beta \cdot \log n) \cdot \mathcal{O}(P_1) \leq$$
$$\leq O(\beta \cdot \log n) \cdot \mathcal{O}(P_\beta).$$

□

Our last heuristic in the context of unequal call bandwidth, $H4$, for problem $CP3$, relates to $H2$ as $H3$ relates to $H1$. It is obtained from Figure 2 by replacing line {*} by:

let $c_i \in C$ be call s.t. $\frac{\sum_{e_k \in r_i} \max(c_i - w_j, 0)}{\sum_{e_k \in r_i} \min(w_k, b_i)}$ is minimal;

We leave further details for the full version of the paper.

3 Competitive Analysis of Call Preemption

In this section we study the call preemption problem in an *on-line* fashion. Generally speaking, an on-line problem is one in which an algorithm that solves it must handle a sequence of requests, satisfying each request without knowledge of the future requests. Examples of on-line problems include the scheduling of jobs in (parallel) machines, paging—i.e., the allocation of cache memory, network routing, maintaining dynamic data structures, etc.

We evaluate on-line algorithms in terms of their *competitiveness* [ST]. A competitive algorithm has the property that its performance on *any* sequence of requests is within a constant factor of the performance of any other algorithm on the same sequence, including the *off-line* algorithm for the problem—which can see all future requests. The study of competitive algorithms is interesting for several reasons. For instance, efficient competitive algorithms have been discovered for a surprising variety of real problems [MMS]. Also, the virtue of an efficient competitive algorithm is that no assumptions need to be made about the distribution of the requests, because it works well for any distribution, and experiments with some competitive algorithms suggest that the competitiveness of an algorithm is a better indication of how well it will perform in practice than average or worst-case analysis [BM].

As the starting point of our study of the competitiveness of a call preemption algorithm, we concentrate on the case comprising two nodes, connected by a channel with capacity k, admitting calls requiring capacity 1. As before, the goal is to come up with a sensible preemption policy such that the disruption created by the preemption process is minimized. But now we will be comparing the performance achieved by that policy to that of the best off-line policy. For a call preemption algorithm A, we define U^A to be the utlization of the channel over a given time period T, and use this as the measure of the algorithm's performance.[2] More specifically, we will call a call preemption algorithm A *c-competitive* if there is a constant c such that for *every* call sequence σ occurring during T

$$U^A(\sigma) \geq \frac{U^*(\sigma)}{c},$$

where U^* is the maximum achievable utilization by any algorithm—including the off-line—for σ. If A is a *randomized* call preemption algorithm, then we say A is c-competitive if $\overline{U^A}(\sigma) \geq U^*(\sigma)/c$ holds, where $\overline{U^A}$ is the utilization given by algorithm A on sequence σ averaged over all the random choices that A makes while processing σ.

On-line algorithms can be viewed as a game against an *adversary*, who is allowed to determine the requests that are incrementally revealed to the algorithm. One can therefore express impossibility arguments in terms of a strategy for the adversary, whose task is to force the competitive ratio to be as large as possible (or, equivalently, a small as possible utilization). It is also customary to distinguish—in the case of randomized on-line algorithms—between *adaptive* and *oblivious* adversaries. The former type would know in advance not just the preemption algorithm, but also the result of the coin tosses performed by the algorithm, while the latter would only know the algorithm [BBK+].

3.1 Unknown Holding Times

We now consider the case where there is no knowledge about the calls' behavior—in particular, the holding time of the calls is not known in advance. Corroborating intuition, we show that in this case it is not possible to design a competitive call preemption algorithm. Let a call c_i be given by the tuple $< a_i, d_i >$, where a_i and d_i indicate as before the call's arrival and holding times, respectively (we drop b_i and r_i from the characterization of the call for being irrelevant in this section).

Proposition 3.1 *If the calls' holding times are not known at arrival time, then there is no deterministic c-competitive call preemption algorithm, for any c.*

Proof: [Sketch] We consider the case $k = 2$, and leave the generalization to any k to the reader. The sequence of calls σ picked by the adversary consists of 3 calls—one "long" call and two "short" calls: $c_1 =< 0, T >$, $c_2 =< 0, 2 >$ and $c_3 =< 1, 1 >$. When (any deterministic) algorithm A receives the first two requests, it allocates one in each of the lines. However, when c_3 arrives, since the holding times are

[2] We note that the choice of this measure assumes no penalty for the preemption of calls. We elaborate more on this point later.

unknown, A chooses c_1, the long call, for preemption. This scenario can be orchestrated by the adversary, who knows A's decisions concerning allocation and preemption of calls. The utilization achieved by A is $U^A = 4$.

On the other hand, the off-line algorithm, able to see the future, chooses c_2, the short call, for preemption. This yields $U^* = T + 2$ and, therefore, an unbounded c. □

The essence of the above reasoning is that the adversary can force the preemption algorithm to preempt the long calls. Moving to randomized algorithms, we note that an adaptive adversary can force the preemption algorithm to do as poorly as the deterministic algorithm. However, one would hope that a randomized preemption algorithm working against an oblivious adversary might perform better, by preempting the short calls with significant probability. We now show that this is not the case.

Proposition 3.2 *If the calls' holding times are not known at arrival time, then there is no randomized c-competitive call preemption algorithm, for any c.*

Proof: [Sketch] (The structure of the proof follows that of [SWW] for randomized scheduling algorithms.) We say a call preemption algorithm is *reasonable* if it never preempts an existing call unless the number of existing calls is k (i.e., the channel is is being fully utilized). We first state some technical lemmas:

Lemma 3.1 *For any unreasonable call preemption algorithm A, there is a reasonable algorithm A' whose worst-case expected performance is at least as good as that of A.*

Lemma 3.2 *The competitive ratio of a reasonable randomized algorithm A can be no less than that of the reasonable algorithm that picks the call to be preempted uniformly.*

The previous lemma shows that it is sufficient to focus on the algorithm that makes its decisions for preemption uniformly. We now show that, same as in the deterministic case, this algorithm can also be "defeated" by the adversary.

The adversary's general strategy will now be to produce "bursts" of short calls to force the uniform algorithm to preempt whatever long calls might exist. More specifically, the adversary picks a set of $k-1$ calls of duration T, and then it produces a burst of short calls. The first call will occupy the only free line, and the remaining will proceed to preempt the long calls. As this process gets repeated, and although it gets more difficult from the adversary's standpoint as the number of remaining long calls decreases, the probability of remaining calls that are long vanishes exponentially for a large enough T. On the other hand, and same as in the deterministic case, the off-line algorithm will always select for preemption the short calls, yielding $U^* \geq (k-1)T$. □

3.2 Known Holding Times

We now show that when the calls' holding times are known, then we can do much better than in the previous subsection—in fact, we are able to show that a 1-competitive algorithm does exist. Although frequently the holding time is not known at the time a call arrives, we believe this result to be relevant since in many circumstances one might be able to provide some estimate of this parameter, depending, for instance, on the type of the call (e.g., voice, video, etc.). Also, a competitive algorithm should be able to exhibit a *learning* behavior in the sense that if there is some structure in the request sequence, then the algorithm should be able to eventually discover this fact and take advantage of it.

In this context, for a call c_i given by $< a_i, d_i >$, let G (for *greedy*) be the deterministic preemption algorithm that chooses for preemption at time t call $c_i \in C$ s.t. $(d_i - (t - a_i))$ is minimal (i.e., the existing call with the least remaining holding time), and chooses arbitrarily in case of a tie.

Proposition 3.3 *If the holding times of the calls are known at arrival time, then G is 1-competitive.*

Proof: [Sketch] Without loss of generality, we assume that no two calls arrive at the same time, and thus we can uniquely order the arrivals. For all calls $c_i \in C$, let $t_i = d_i - (t - a_i)$ (i.e., t_i is the remaining time of call c_i at time t). We will represent the status of the channel at time t by the tuple $\tau(t) =< t_{i_1}, t_{i_2}, ..., t_{i_k} >$, such that $t_{i_j} \leq t_{i_{j+1}}$, for $1 \leq j \leq k-1$. We say that channel status tuple $\tau(t) \succeq \tau'(t)$ (we read $\succeq$ as "dominates") iff $t_{i_j} \geq t'_{i_j}$, for $1 \leq j \leq k$.

After each call arrival and preemption takes place, we reorder the tuple's components such that the non-decreasing order is maintained. The claim follows from the fact that algorithm G yields after each call arrival a status tuple that dominates that produced by any other preemption algorithm (including the off-line algorithm). I.e., $\tau^G(t) \succeq \tau^A(t)$, for any preemption algorithm A. $\square$

We note that Proposition 3.3 has an impact on an alternate preemption policy in which arriving calls are only accepted if they would yield higher utilization (in this case, an arriving call will be accepted if there exists a call with remaining holding time less than the arriving call's). Also, and perhaps more realistically, if instead of the exact holding times only the probability distribution of the calls' holding times is known, then, based on G, the best (expected) competitiveness is achievable.

4 Conclusions

In this paper we have addressed the problem of preempting ongoing calls in a communication network in order to accommodate new calls. To the best of our knowledge, no analytical study of call preemption has been published.

We have shown that versions of the problem are computationally intractable, and provided simple and efficient heuristics to approximate the optimal strategy. We have also studied the problem from the relatively new *on-line* perspective, and characterized what can be done under different circumstances. We note that in our study we assumed no penalty for the preemption of calls. We already have some preliminary results for the case when penalty exists; this will be the subject of a future paper.

Acknowledgements

The authors would like to thank Israel Cidon for his contributions in formulating the problem. We are also thankful to Shay Kutten for useful suggestions in early stages of this work, Baruch Schieber for illuminating discussions on on-line algorithms, and Moti Yung for his part in approximation results.

References

[ATM] "Special Issue on Asynchronous Transfer Mode," *Int. Journal of Digital and Analog Cabled Systems*, 1(4), 1988.

[BBK+] S. Ben-David, A. Borodin, R. Karp, G. Tardos and A. Wigderson, "On the power of randomizetion in on-line algorithms," *Proc. 22th ACM Symp. on Theory of Computing*, pp. 379-386, May 1990.

[BM] J. Bentley and C. McGeoch, "Amortized analyses of self-organizing sequential search heuristics," *Communications of the ACM*, 28(4), pp. 404-411, April 1985.

[CDF+] D. Clark, B. Davie, D. Farber, I. Gopal, B. Kadaba, W. Sincoskie, J. Smith and D. Tennenhouse, "The AURORA Gigabit Testbed," to appear in *Computer Networks.*

[CG] I. Cidon and I. Gopal, "PARIS: An approach to integrated high-speed private networks," *Int. Journal of Digital and Analog Cabled Systems*, 1(2), pp. 77-86, 1988.

[D] D. Johnson, "Algorithms for Combinatorial Problems," *Journal of Computer and Systems Sciences*, 9, pp. 256-278, 1974.

[GJ] M. Garey and D. Johnson, *Computers and Intractability, a Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, New York, 1979.

[MMS] M. Manasses, L. McGeoch and D. Sleator, "Competitive Algorithms for On-line Problems," *Proc. 20th ACM Symp. on Theory of Computing*, pp. 322-333, May 1988.

[ST] D. Sleator and R. Tarjan, "Amortized efficiency of list update and paging rules," *Communications of the ACM*, 28(2), pp. 202-208, 1985.

[SWW] D. Shmoys, J. Wein and D. Williamson, "Scheduling Parallel Machines On-line," *Proc. 32nd IEEE Symp. on Found. of Computer Science*, pp. 131-140, October 1991.